Home

Axolotl Ratchet

Goals

- Combine the forward-secrecy of symmetric-key updating with the "future-secrecy" of an OTR-like Diffie-Hellman ratchet.
  - By "future-secrecy" we mean that a leak of keys to a passive eavesdropper will be healed by introducing new DH ratchet keys.
- Improve on OTR's future-secrecy with a "2-step" DH ratchet instead of "3-step".
  - I.e. DH ratchet keys are used in a "send/receive" pattern, instead of "advertise/receive/send".
- Detect replay / reorder / deletion of messages.
- Allow decryption of out-of-order messages with minimal reduction in forward secrecy.
- Don't leak metadata in cleartext (such as identities or sequence numbers).

Algorithm

```
State
------
Each party stores the following values per conversation in persistent storage:

RK            : 32-byte root key which gets updated by DH ratchet
HKs, HKr      : 32-byte header keys (send and recv versions)
NHKs, NHKr    : 32-byte next header keys (")
CKs, CKr      : 32-byte chain keys (used for forward-secrecy updating)
DHIs, DHIr    : DH or ECDH Identity keys
DHRs, DHRr    : DH or ECDH Ratchet keys
Ns, Nr        : Message numbers (reset to 0 with each new ratchet)
PNs           : Previous message numbers (# of msgs sent under prev ratchet)
ratchet_flag  : True if the party will send a new DH ratchet key in next msg
skipped_HK_MK : A list of stored message keys and their associated header keys
                for "skipped" messages, i.e. messages that have not been
                received despite the reception of more recent messages.
                Entries may be stored with a timestamp, and deleted after a
                certain age.


Key Agreement
--------------
 - Parties exchange identity keys (A, B) and handshake keys (A0, A1) and (B0, B1)
 - Parties assign themselves "Alice" or "Bob" roles by comparing public keys
 - Parties calculate master key using tripleDH:
   - master_key = HASH( DH(A, B0) || DH(A0, B) || DH(A0, B0) )

Alice:
  KDF from master_key: RK, HKs=<none>, HKr, NHKs, NHKr, CKs=<none>, CKr
  DHIs, DHIr = A, B
  DHRs, DHRr = <none>, B1
  Ns, Nr = 0, 0
  PNs = 0
  ratchet_flag = True
Bob:
  KDF from master_key: RK, HKr=<none>, HKs, NHKr, NHKs, CKr=<none>, CKs
  DHIs, DHIr = B, A
  DHRs, DHRr = B1, <none>
  Ns, Nr = 0, 0
  PNs = 0
  ratchet_flag = False


Sending messages
-----------------
Local variables:
  MK  : message key

if ratchet_flag:
  DHRs = generateECDH()
  HKs = NHKs
  RK, NHKs, CKs = KDF( HMAC-HASH(RK, DH(DHRs, DHRr)) )
  PNs = Ns
  Ns = 0
  ratchet_flag = False
MK = HMAC-HASH(CKs, "0")
msg = Enc(HKs, Ns || PNs || DHRs) || Enc(MK, plaintext)
Ns = Ns + 1
CKs = HMAC-HASH(CKs, "1")
return msg


Receiving messages
-------------------
Local variables:
  MK  : message key
  Np  : Purported message number
  PNp : Purported previous message number
  CKp : Purported new chain key
  DHp : Purported new DHr
  RKp : Purported new root key
  NHKp, HKp : Purported new header keys

Helper functions:
  try_skipped_header_and_message_keys() : Attempt to decrypt the message with skipped-over
  message keys (and their associated header keys) from persistent storage.

  stage_skipped_header_and_message_keys() : Given a current header key, a current message number,
  a future message number, and a chain key, calculates and stores all skipped-over message keys
  (if any) in a staging area where they can later be committed, along with their associated
  header key.  Returns the chain key and message key corresponding to the future message number.

  commit_skipped_header_and_message_keys() : Commits any skipped-over message keys from the
  staging area to persistent storage (along with their associated header keys).

if (plaintext = try_skipped_header_and_message_keys()):
  return plaintext

if HKr != <none> and Dec(HKr, header):
  Np = read()
  CKp, MK = stage_skipped_header_and_message_keys(HKr, Nr, Np, CKr)
  if not Dec(MK, ciphertext):
    raise undecryptable
else:
  if ratchet_flag or not Dec(NHKr, header):
    raise undecryptable()
  Np = read()
  PNp = read()
  DHRp = read()
  stage_skipped_header_and_message_keys(HKr, Nr, PNp, CKr)
  HKp = NHKr
  RKp, NHKp, CKp = KDF( HMAC-HASH(RK, DH(DHRp, DHRs)) )
  CKp, MK = stage_skipped_header_and_message_keys(HKp, 0, Np, CKp)
  if not Dec(MK, ciphertext):
    raise undecryptable()
  RK = RKp
  HKr = HKp
  NHKr = NHKp
  DHRr = DHRp
  erase(DHRs)
  ratchet_flag = True
commit_skipped_header_and_message_keys()
Nr = Np + 1
CKr = CKp
return read()
```

Variations

- Header encryption may be omitted if the underlying transport is already leaking metadata, and space is at a premium.
  - In that case, the presence of a new ratchet key signals the recipient that the DH ratchet is advancing (instead of using encryption by the next header key as the signal).
  - Instead of storing old header keys for skipped messages, old ratchet keys can be used to recognize delayed messages.
- Depending on how the key agreement is performed, it may be possible to omit the A1 and/or B1 keys.
  - If a party knows she is the initiator (Alice) prior to sending her key agreement message, then she doesn't need to send the extra (A1) key, as it is unused.
  - If Bob doesn't plan to send any messages prior to receiving Alice's first message, B1 can be omitted and both parties can set B1 equal to B0 with no loss of security.
- The chain keys could be updated on a time basis as well as a per-message basis.
  - For example: If 24 hours elapse without receiving a message, you might wish to move to the next chain key in case there's an intercepted message you're unaware of.

IPR

The Axolotl specification (this wiki) is hereby placed in the public domain.

Feedback

Can be sent to axolotl at trevp.net

Acknowledgements

Joint work with Moxie Marlinspike.

Thanks to Michael Rogers and Adam Back for mailing list discussions. Adam proposed separating message keys from chain keys. Michael proposed updating keys on a time basis, in addition to a per-message basis.

Thanks to Adam Langley for discussion and improving the receiving algorithm.

Last edited by trevp, 13 days ago